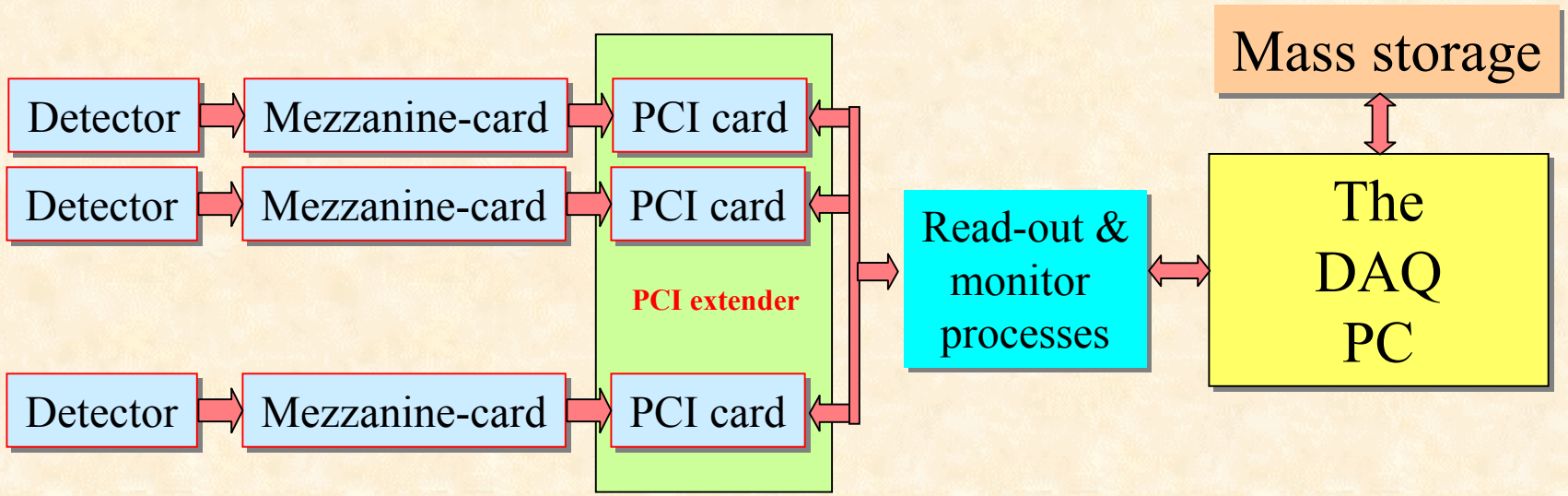
Status of the DAQ design and implementation for the beam test

G. Alimonti, G. Chiodini, S. Magni,
D. Menasce, L. Uplegger

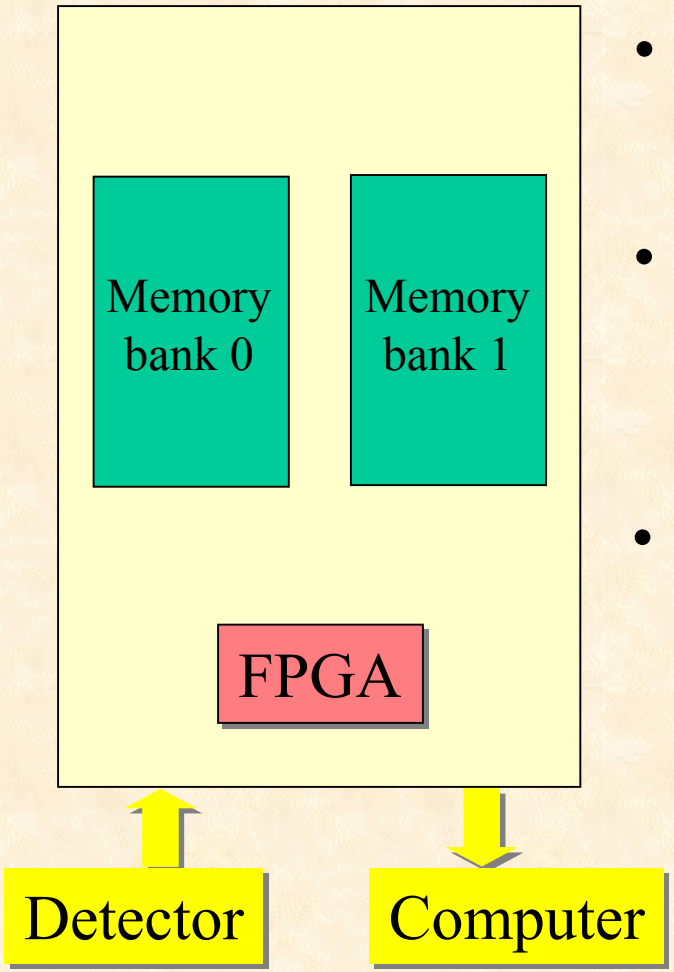
Our aim

- We need to be able to read-out events from detectors placed either on a beam or on a test bench
- We intend to make use of the PCI standard boards developed by the group of Sergio Zimmerman (ESE Feynman)
- A conceptual schematics of the architecture is the following



The PCI card

- A PCI card contains, schematically, three major components:



- The FPGA controls the logic of this board and can be programmed
- Our aim is to provide the software components that allow a read out of these memories to the host DAQ PC.
- This task has two distinct technological aspects:
  - FPGA programming (low level language)
  - read-out of the memory banks and synchronization with the host PC (high level language, C++)

The logic of the PCI board

- When a memory bank in the PCI board is full, the FPGA swaps to the other one and raises an interrupt that is catched by an external process (an Interrupt Handler, IH) to notify the computer that data are ready to be fetched and transfered to an external storage
- Since we will have several PCI boards lodged in a PCI extender, each one getting data from one or more detectors, we need to design an overall architecture to synchronize this transfer activity and allow for a later stage event-building.
- We will take data in an asynchronous mode: each time a pixel has data above threshold, it sends them to the PCI memory bank where they are stored along with row/column and time-stamp information.

And event is thus defined as: all hits labeled by the same time-stamp

The read-out philosophy

- We would like to preserve the ability of getting events also in absence of a trigger: we therefore need to provide a mechanism to restore an event by assembling it's components, marked by the same time-stamp, and we would like to do this in the most simple way possibile (least impact on hardware and software requirements)
- We devised three possibile approaches to this problem (and already developed and tested one of these alternatives):

1. Each PCI board is read-out as soon as data are available, independently from one-another. In this case building an event is a non-trivial feat (we must define clear boundaries in time whithin which to look for an event element). We disfavor this approach
2. We synchronize, by two possible mechanisms, the swapping of memory banks, making thus events somewhat contiguous in the output data stream, and easier to build at a later stage.

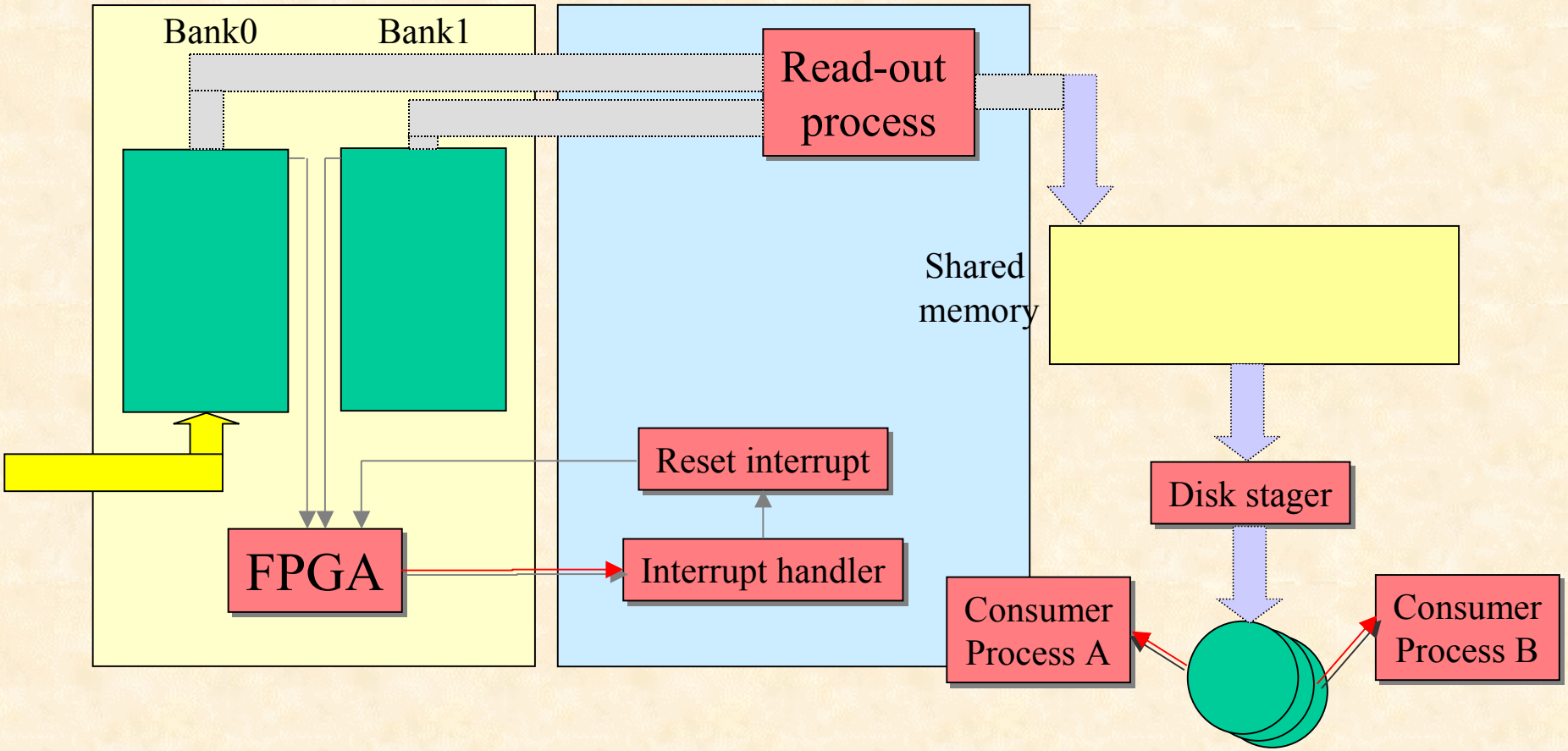
The basic mechanism

- The basic solution we would like to adopt is the one that makes the event-builder the easiest to write, since this is, in our opinion, the approach that provides the most elegant way of controlling the flux of the data from the PCI boards to the host computer.

- I will show in a cartoon the basic mechanism of read-out of a single PCI board (this is easy), but I will only mention the approach we have taken in developing the overall read-out when it comes to synchronize many boards togheter.

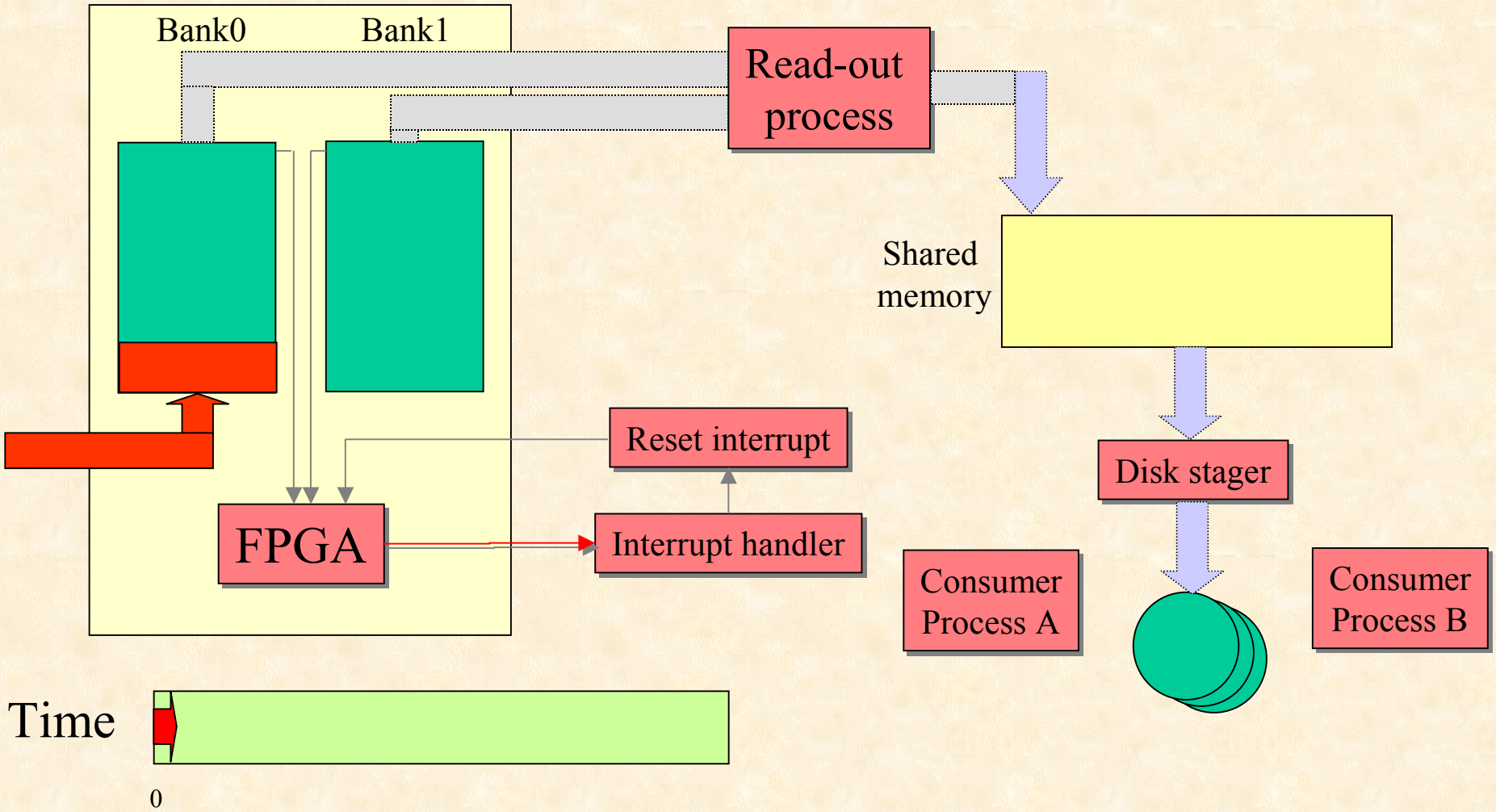
Components (0)

- Basic mechanism of operation of a PCI board and the read-out:



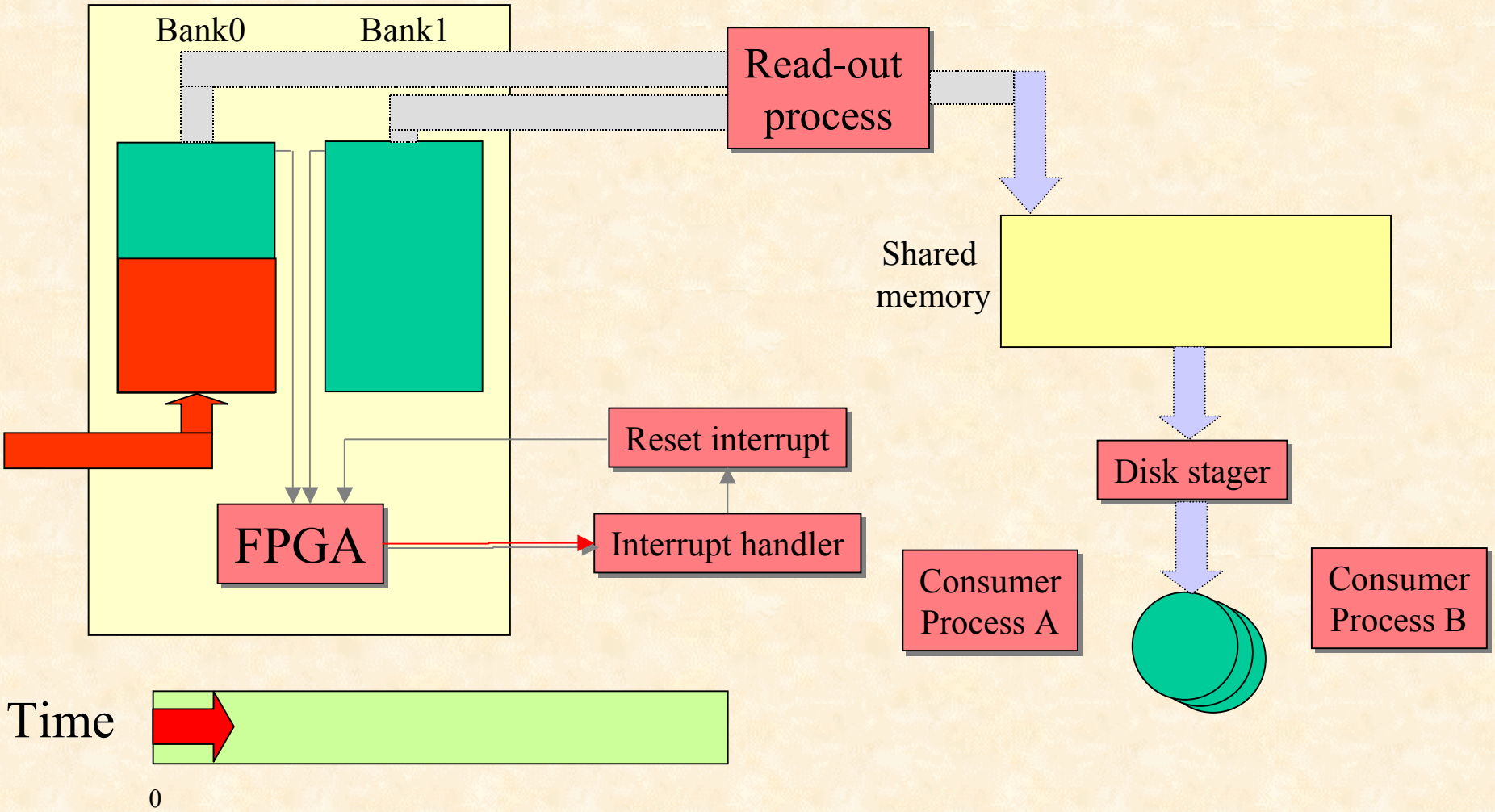
Components (1)

- Basic mechanism of operation of a PCI board and the read-out:



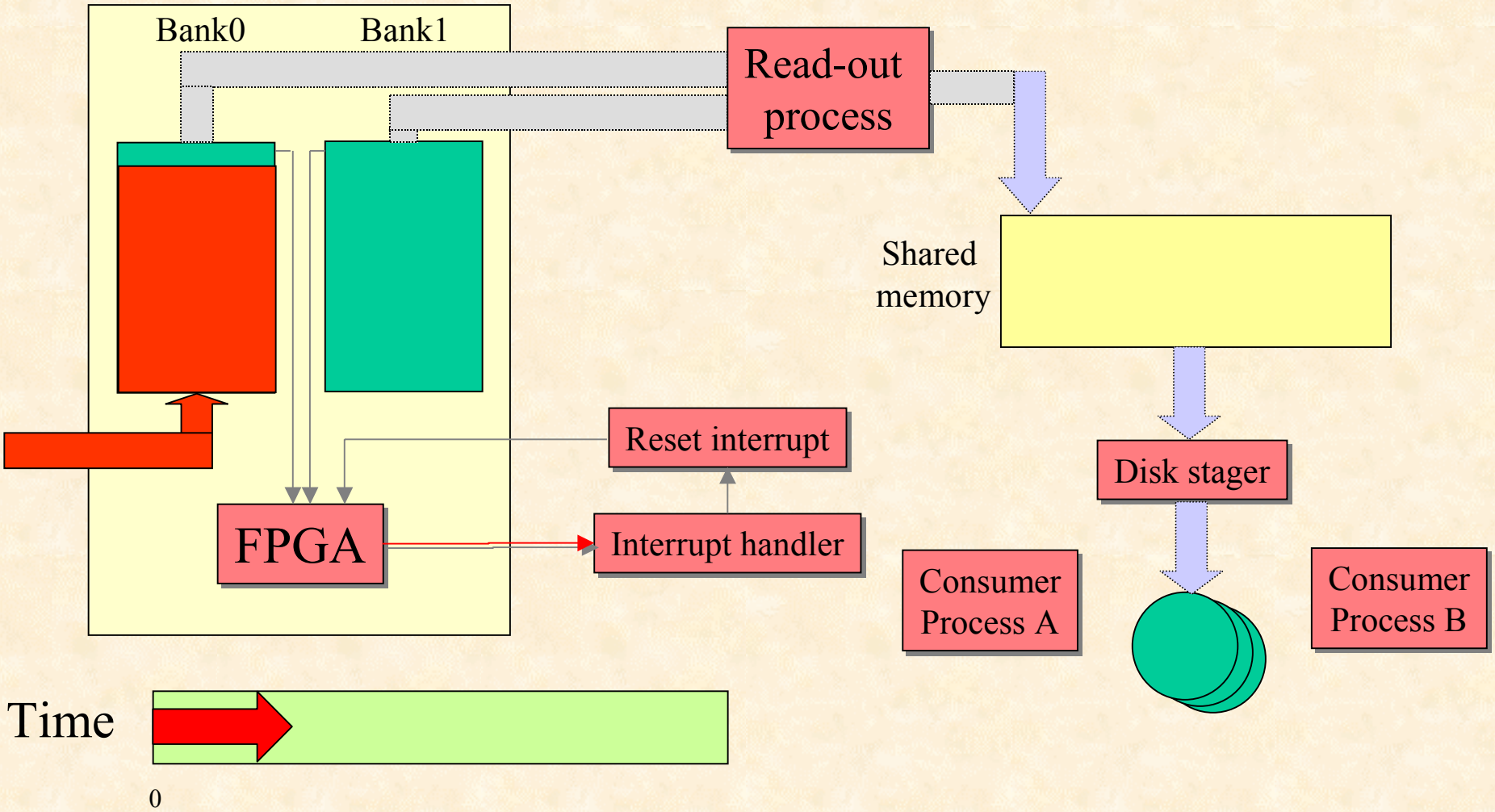
Components (2)

- Basic mechanism of operation of the PCI card and the read-out:



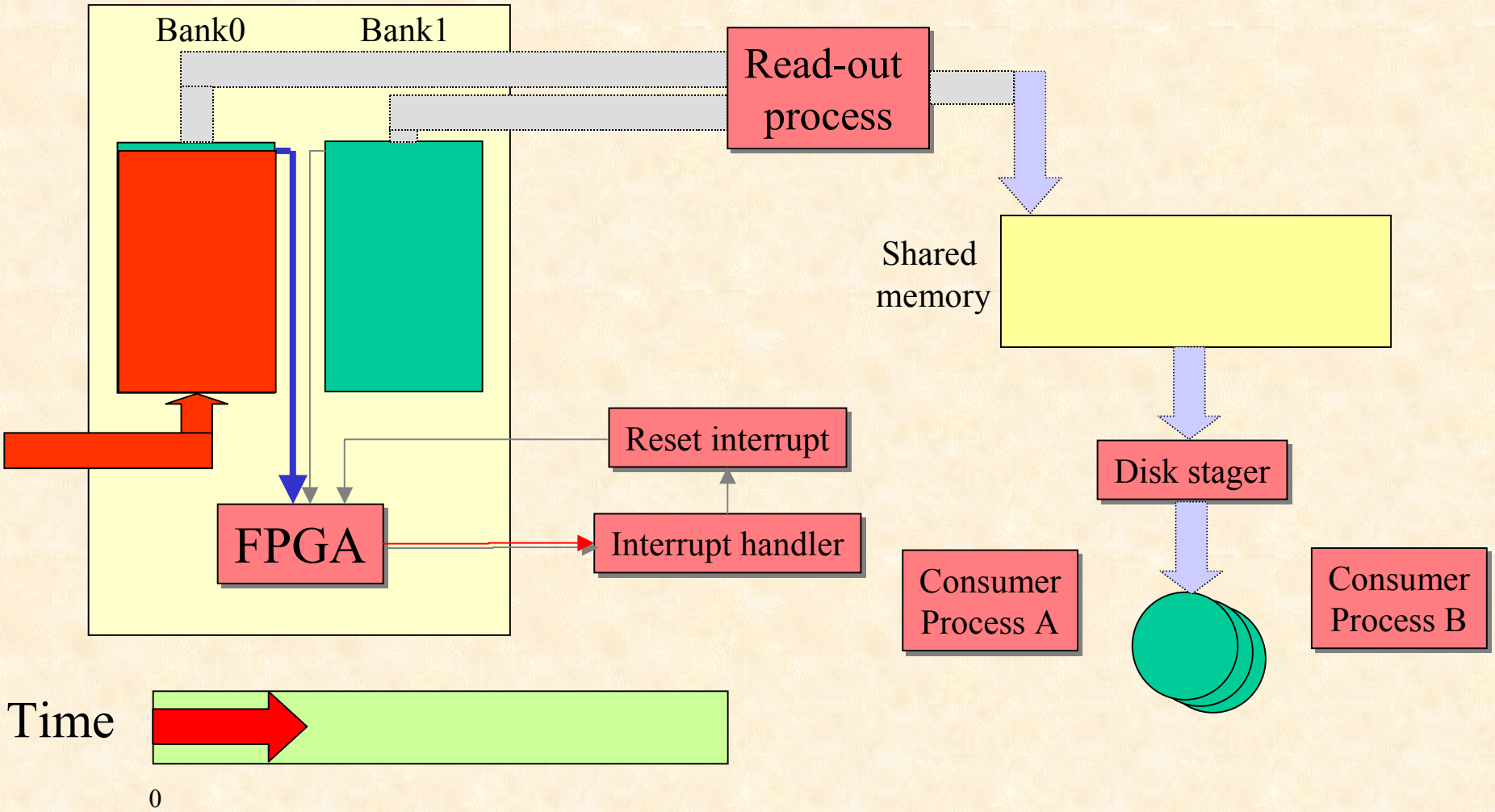
Components (3)

- Basic mechanism of operation of the PCI card and the read-out:



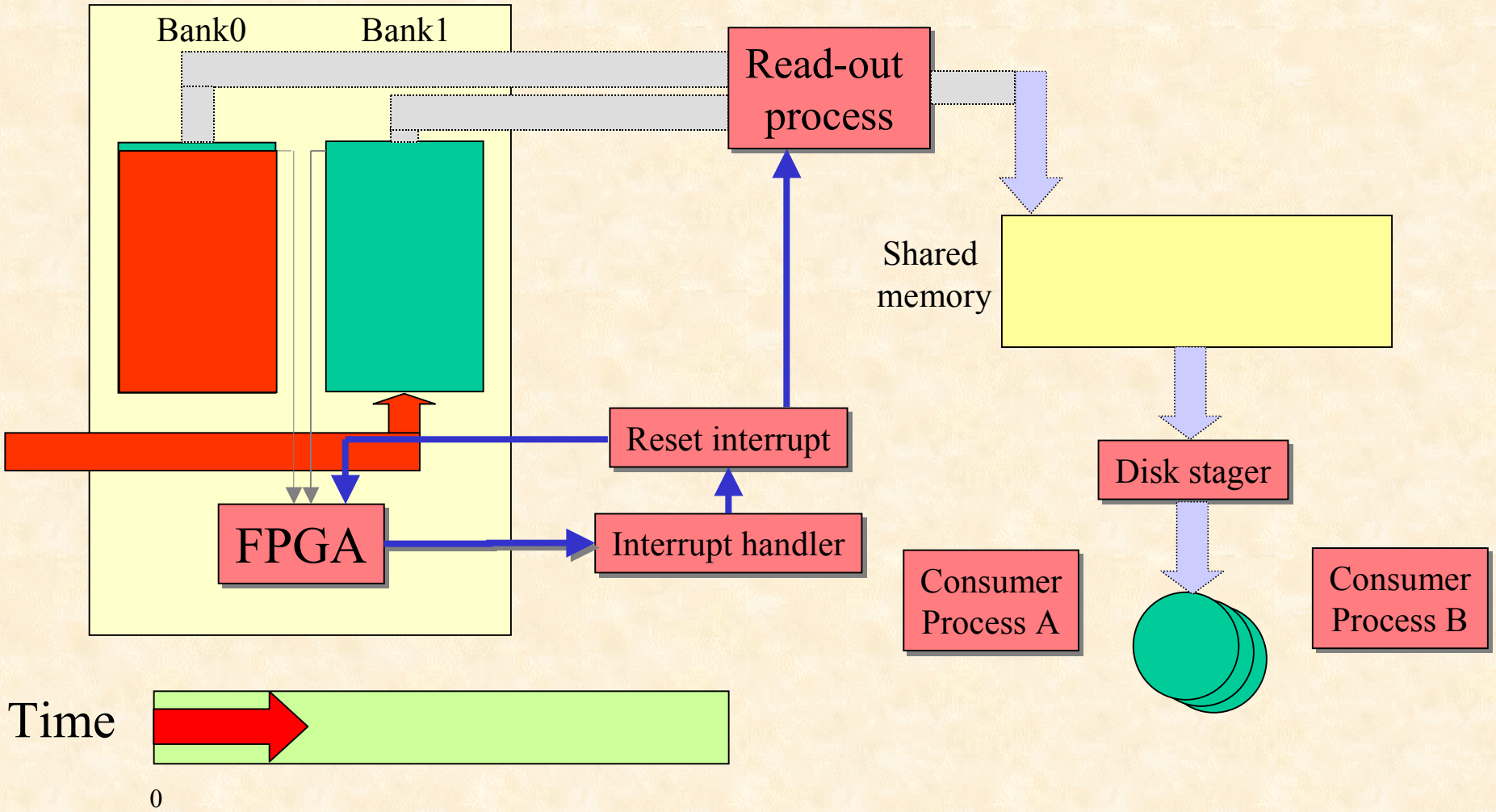
Components (4)

- Basic mechanism of operation of the PCI card and the read-out:



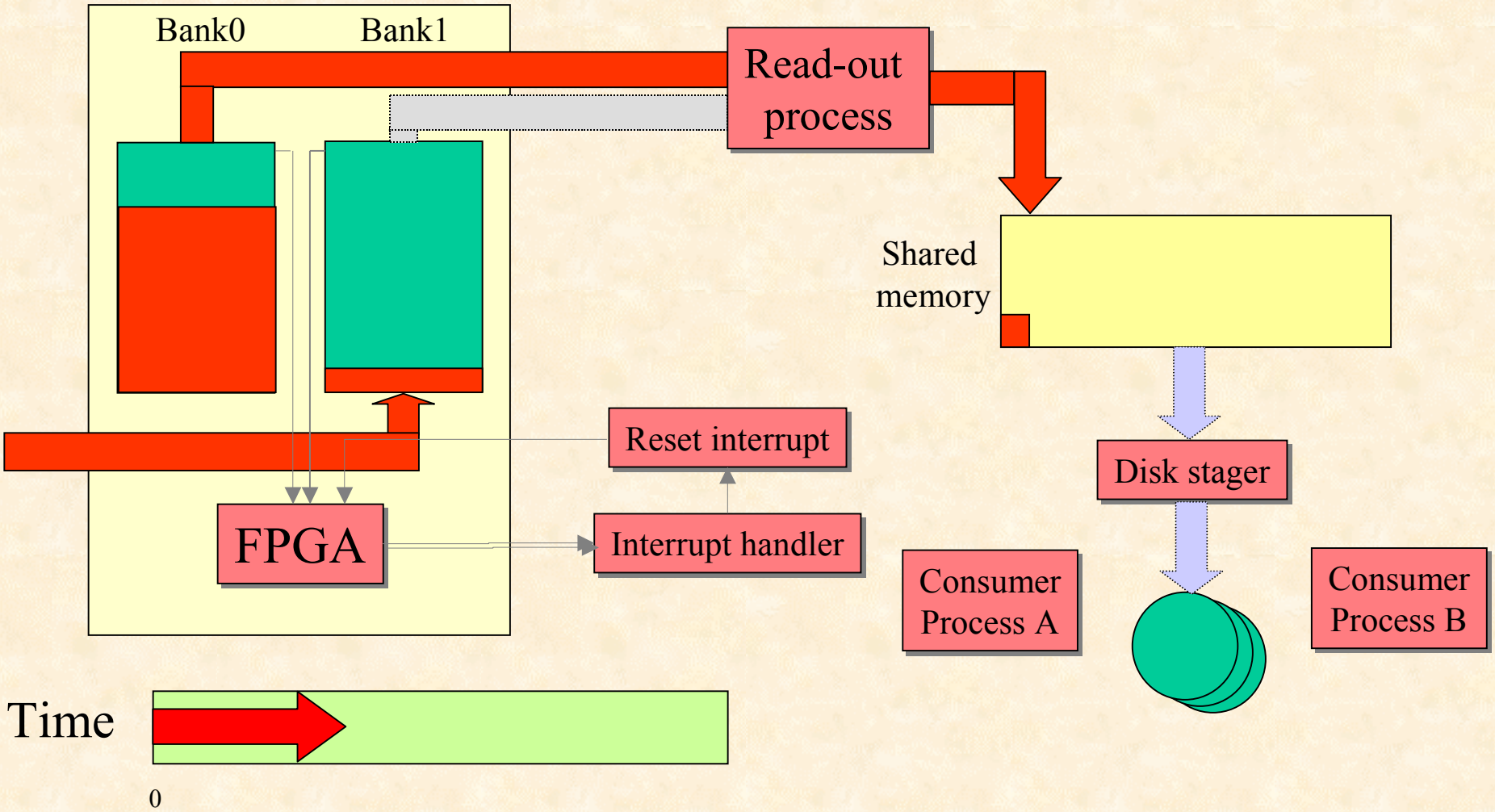
Components (5)

- Basic mechanism of operation of the PCI card and the read-out:



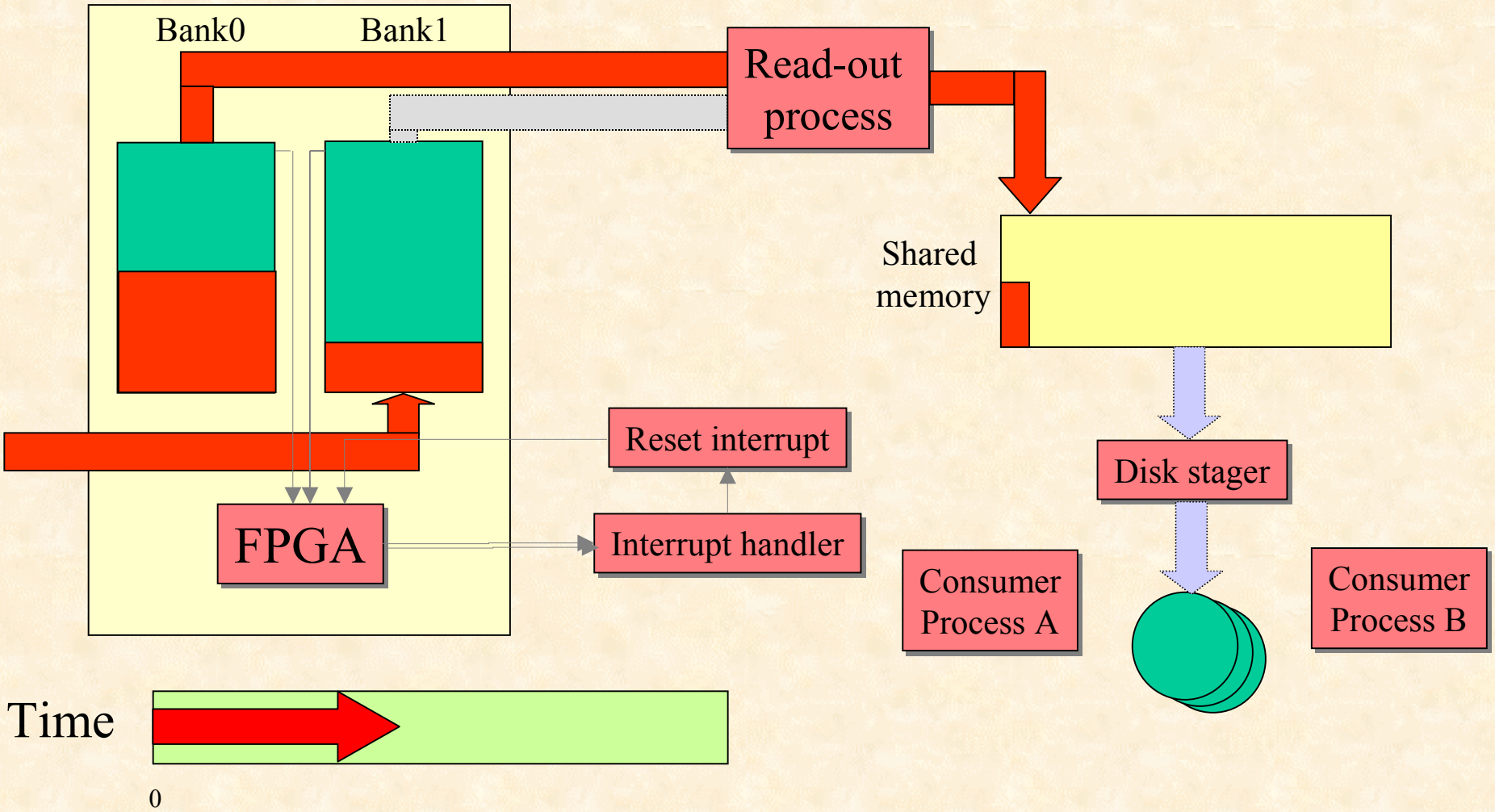
Components (6)

- Basic mechanism of operation of the PCI card and the read-out:



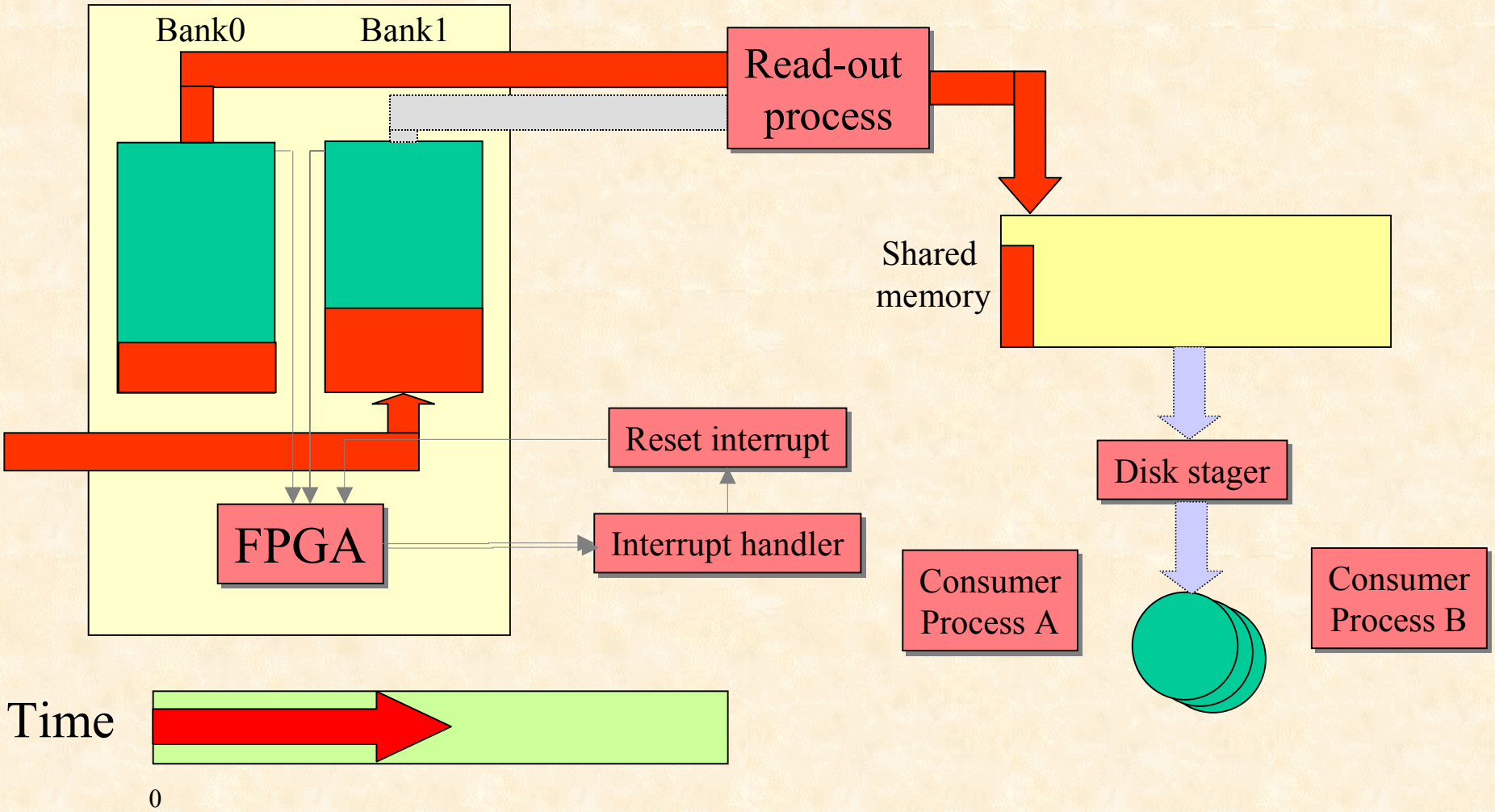
Components (7)

- Basic mechanism of operation of the PCI card and the read-out:



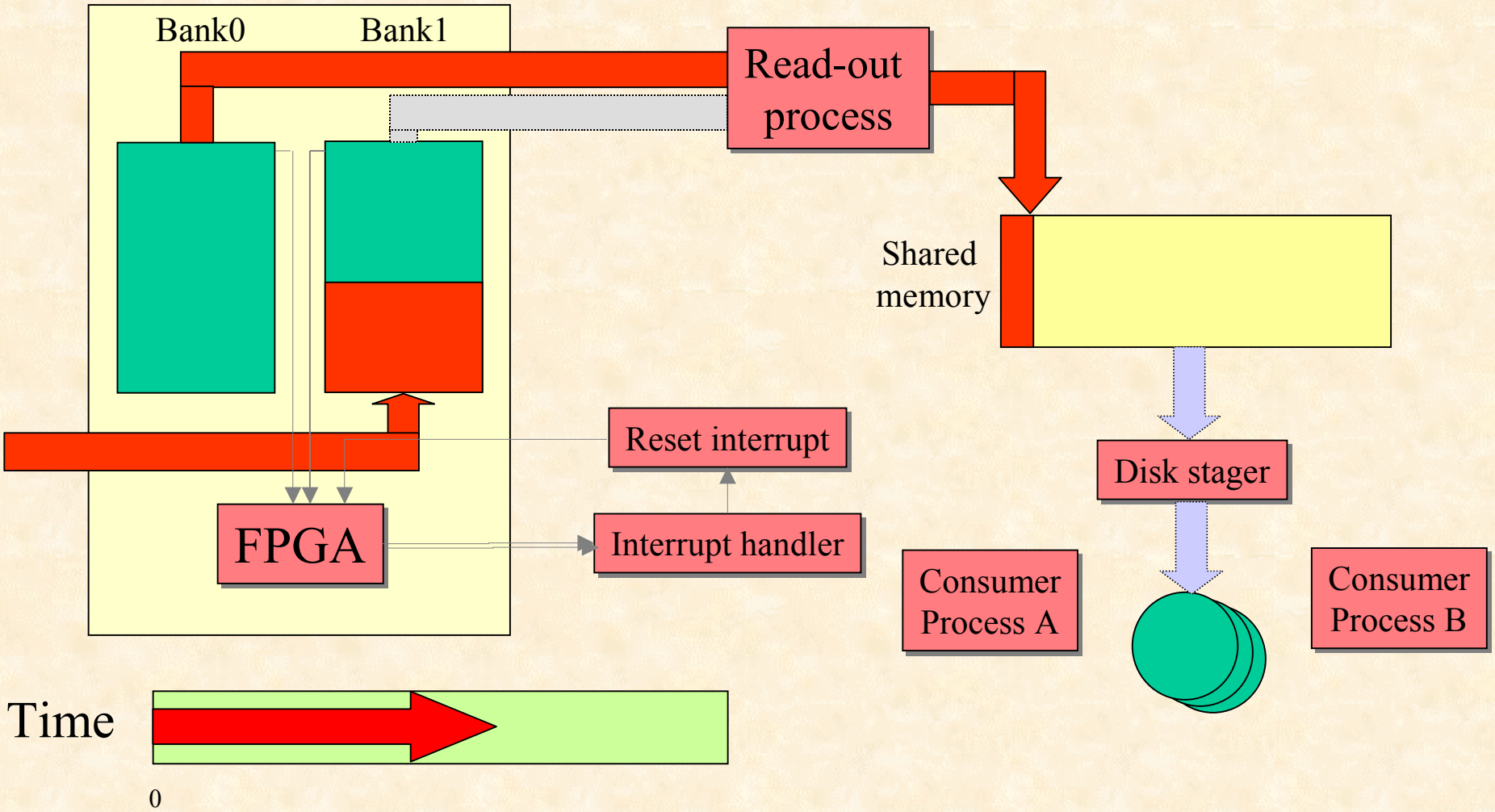
Components (8)

- Basic mechanism of operation of the PCI card and the read-out:



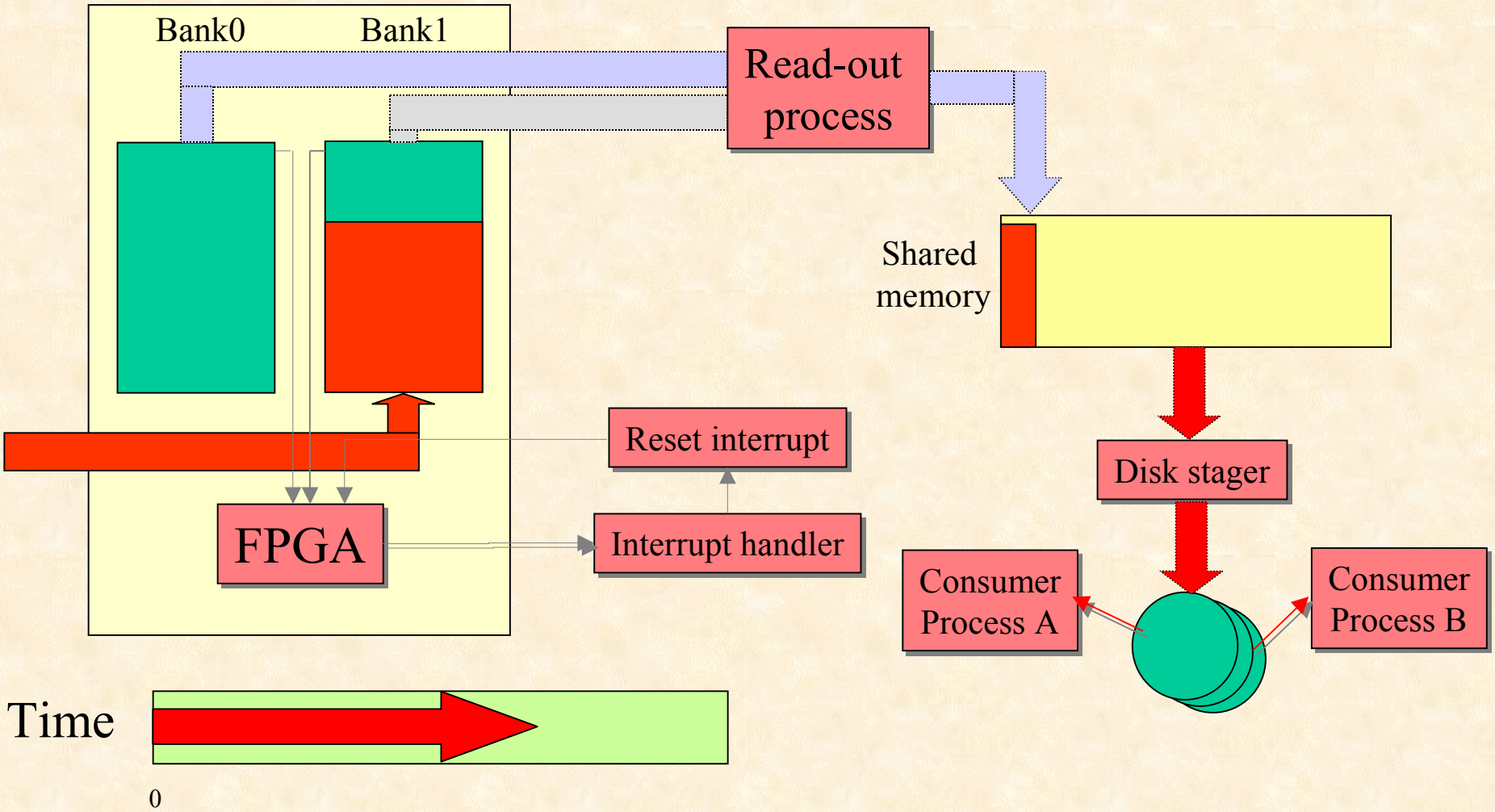
Components (9)

- Basic mechanism of operation of the PCI card and the read-out:



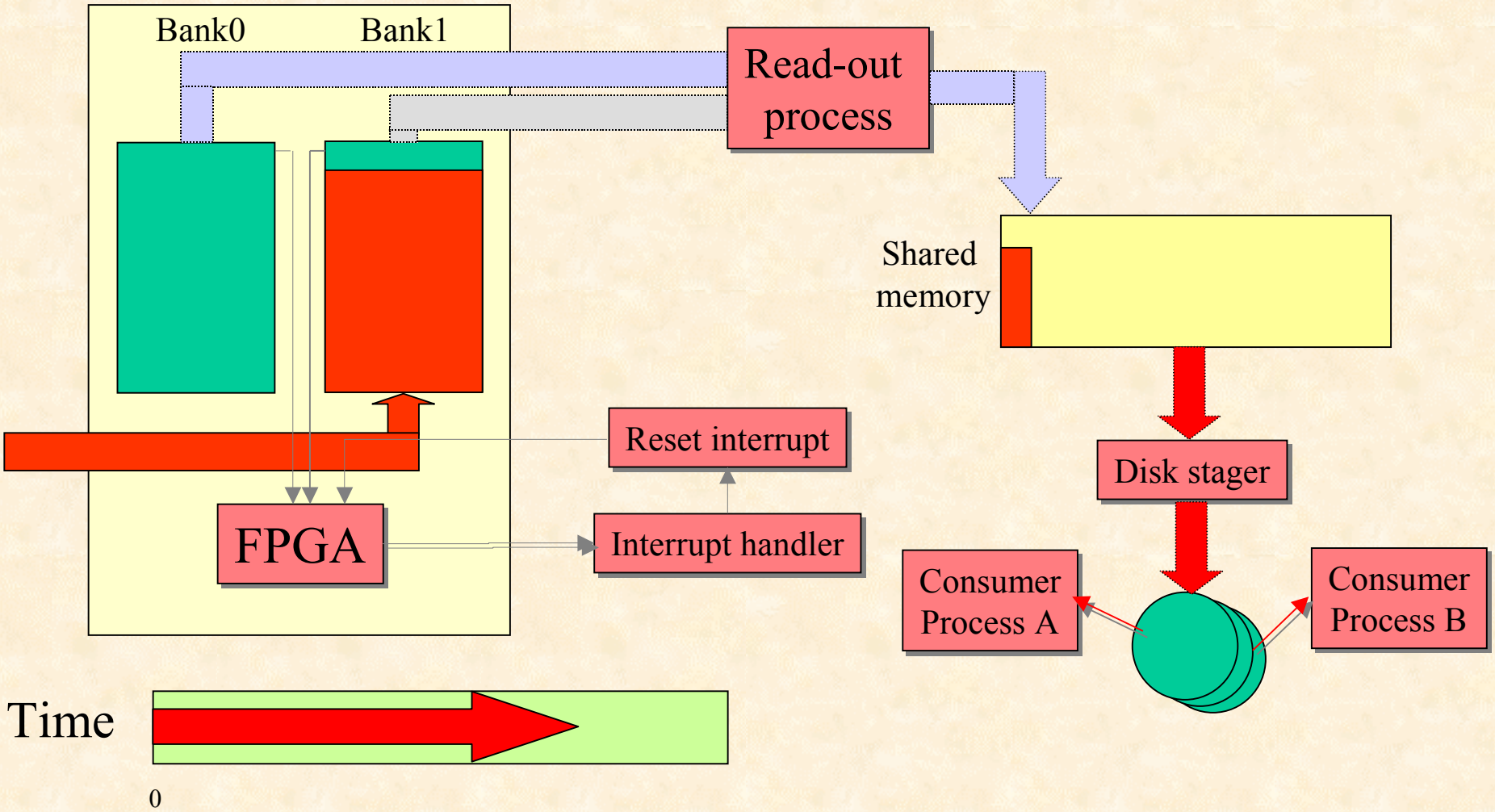
Components (10)

- Basic mechanism of operation of the PCI card and the read-out:



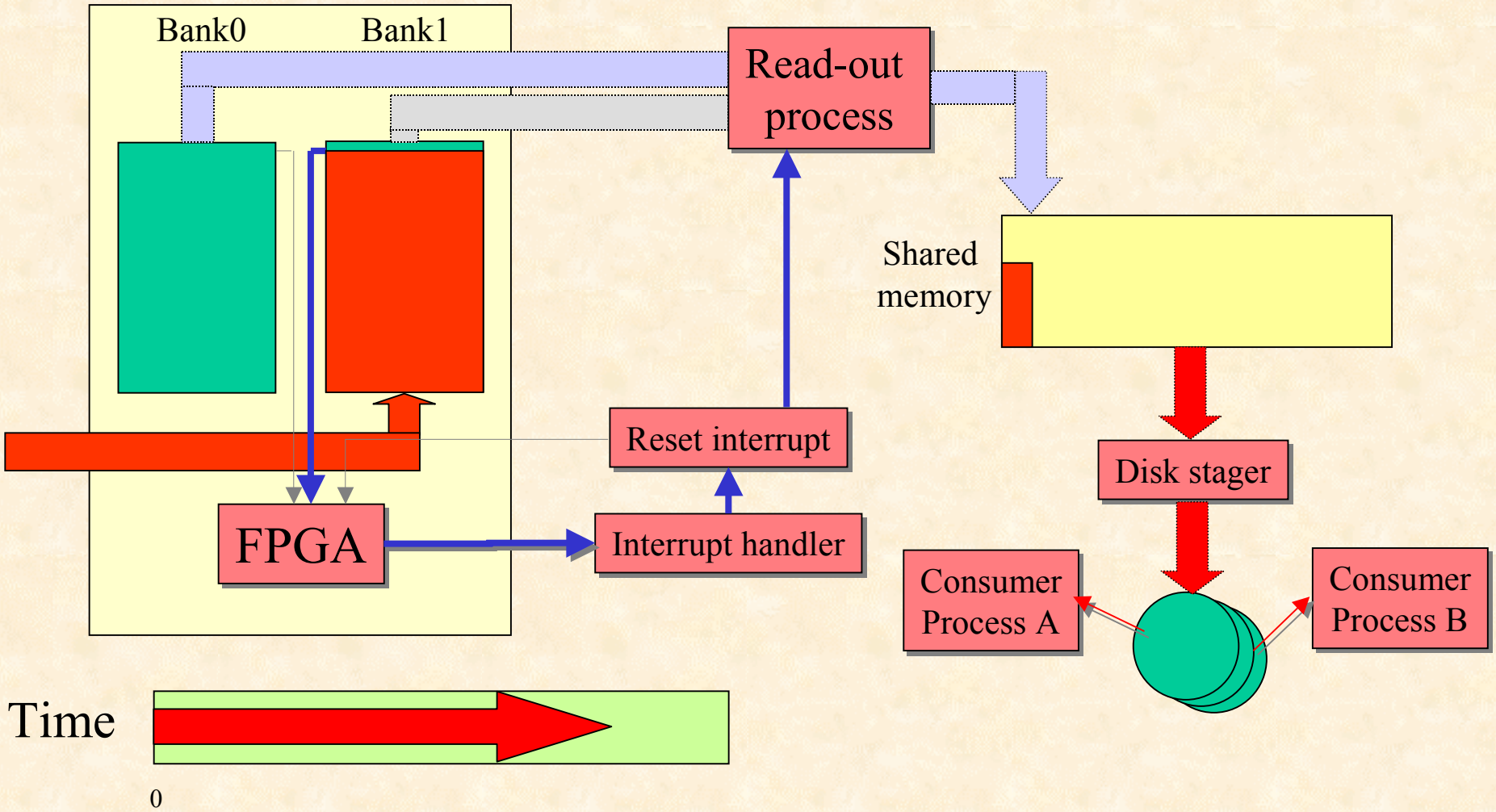
Components (11)

- Basic mechanism of operation of the PCI card and the read-out:



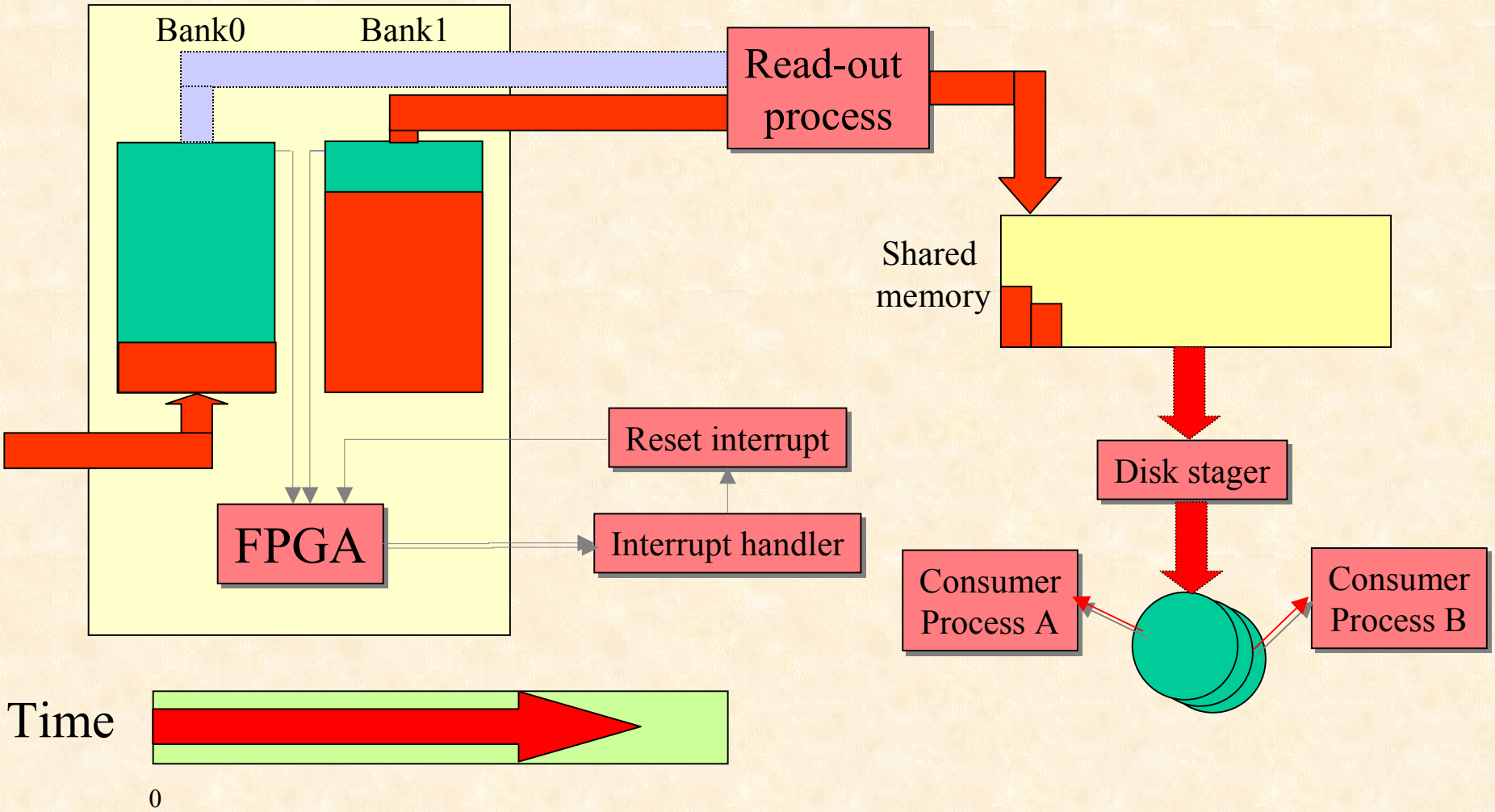
Components (12)

- Basic mechanism of operation of the PCI card and the read-out:



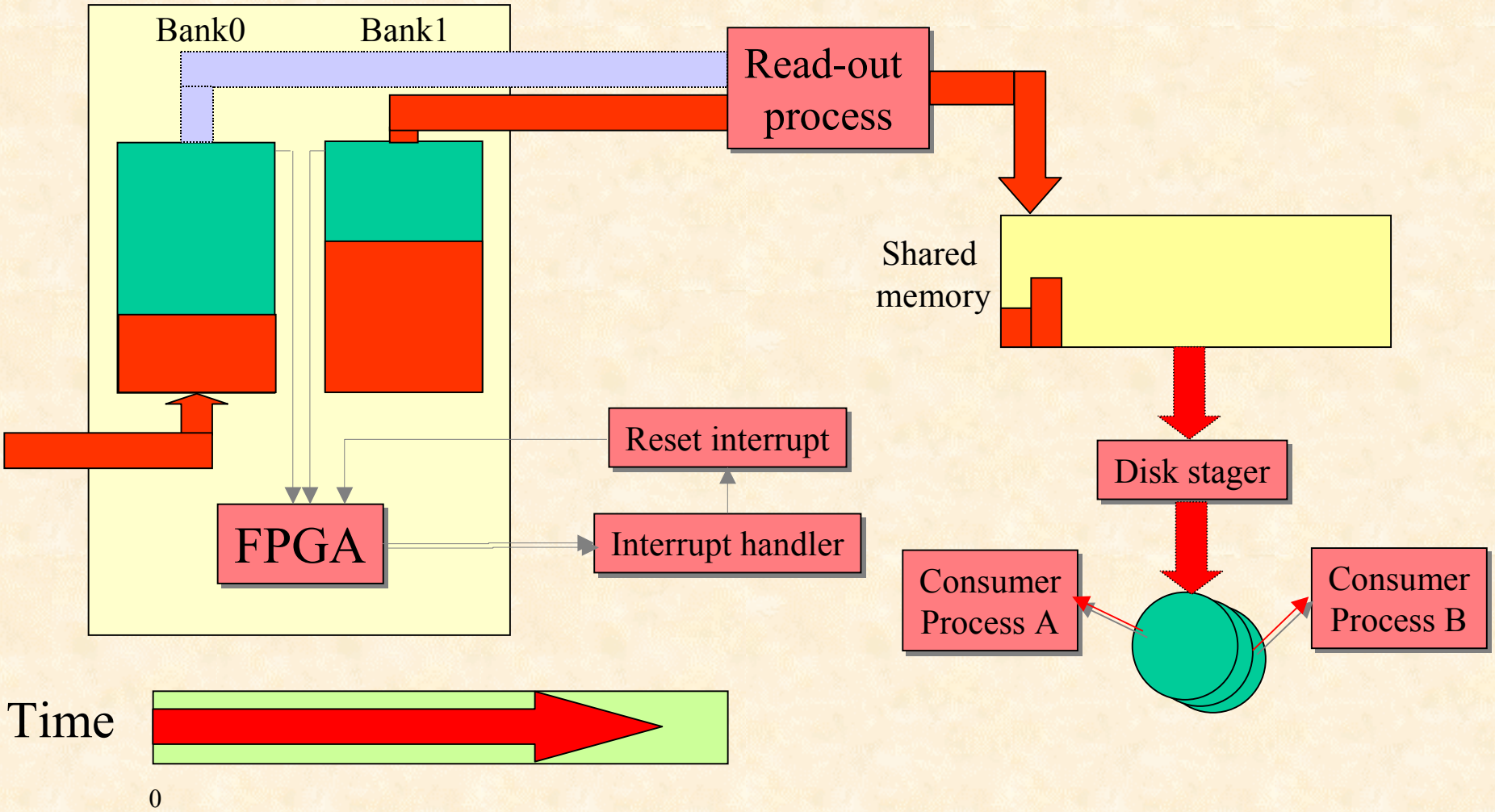
Components (13)

- Basic mechanism of operation of the PCI card and the read-out:



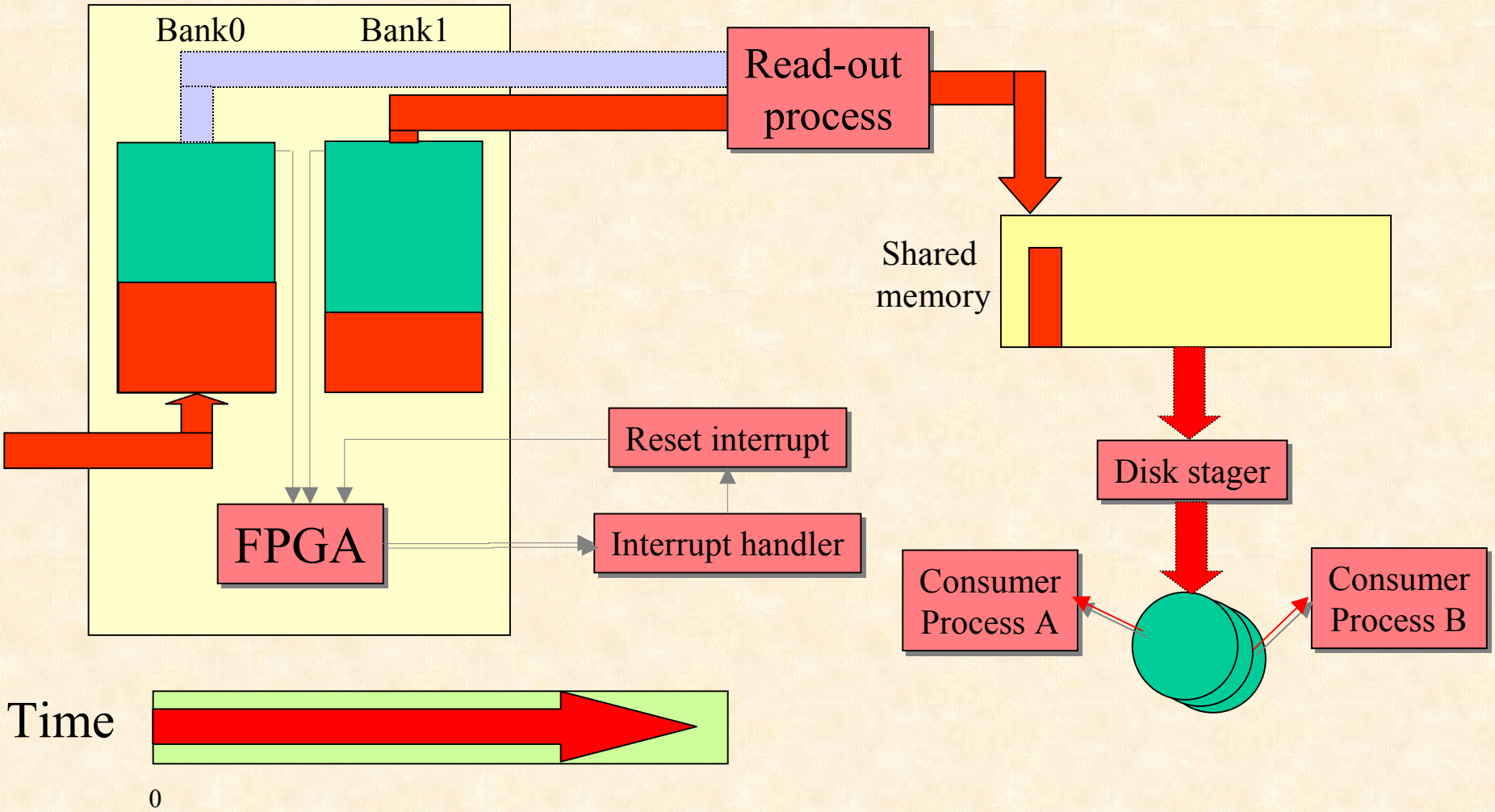
Components (14)

- Basic mechanism of operation of the PCI card and the read-out:



Components (15)

- Basic mechanism of operation of the PCI card and the read-out:



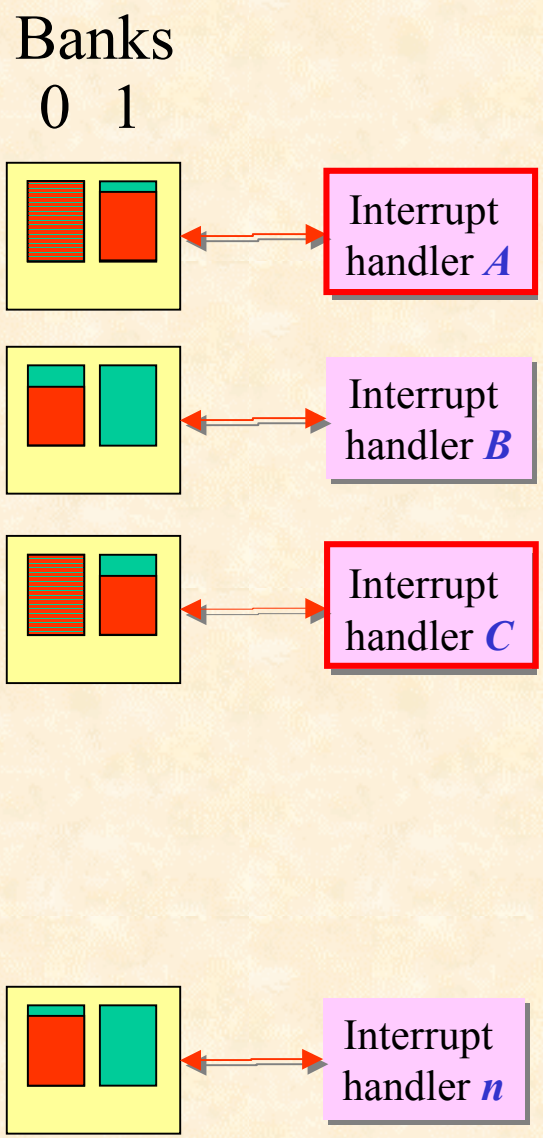
More than one PCI

Problem is, though, that we have more than one PCI board to be read-out: if we read them out independently in time, data related to one event may end up in far away places in the shared memory buffer on the host computer, making the event building a non-trivial task.

The solution we have envisaged and developed is to synchronize the swapping of the memory banks to the first one that gets filled up. Once a bank is full, all banks are forced to swap, no matter what their content status, and the read out can start on them (while the other banks continue to receive data from the detector in a continuous flux).

In this way the shared memory acts as a balancing buffer to compensate for unexpected rate fluctations: if the read-out is synchronized this way, hits with same time-stamp end up, at most, in two different swapping cycles, making the event builder a rather trivial implementation of a sorting algorithm (with boundaries)

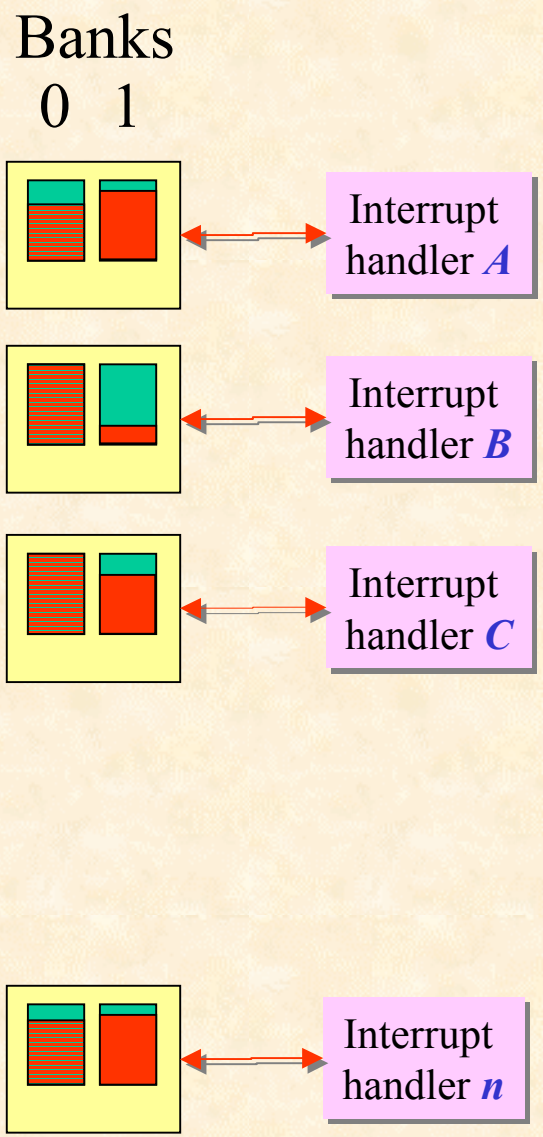
A simple sketch



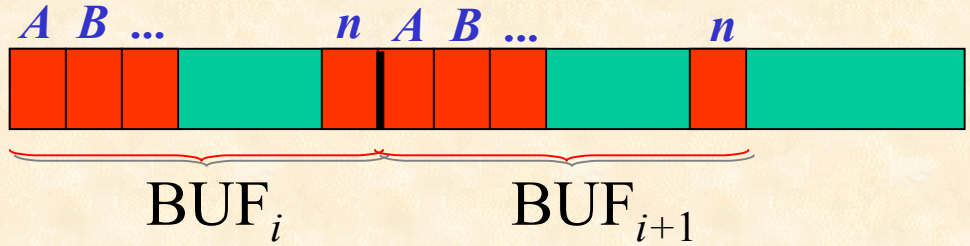
- If each PCI board is read-out as soon as one of its banks is full, hits with same time stamp can end-up anywhere in the read-out shared memory.
- If board ***B*** has few data to send per event, on average, it will be read-out relatively rarely, so hits pertaining to events already written down earlier will be spaced far away in this memory (actually anywhere)

We consider this approach unappealing

The alternative



- If , on the other hand we synchronize the swapping (the clock governing this swapping can be set by the board with the first memory reaching the full status), events with same time-stamp will end up not too far away in the shared memory. Pieces of one event will actually be at most in two adjacent buffers corresponding to two consecutive swap cycles



Problems identified and solved

- This synchronized cyclic swapping needs a strategy to be correctly implemented. Either the first board reaching the full status forces the others to swap (upon appropriate checks that it can do so, in order to avoid conflicting orders that can enact unwanted multiple swaps), or they swap togheter in unison.
- In the first case a complex synchronization among boards must be setup (each board must be somehow knowledgeable about each other's read-out and swapping status). We already tried this solution and found it working satisfactorily.
- The latter option requires an additional hardware component to be placed on the board (actually a cable), connecting all interrupt lines togheter. We strongly favor this approach, and already discussed with the ESE group wheater this can be accomplished shortly.

What has already been accomplished

The architecture mentioned above poses several problems (each board must be knowledgeable about each other's status in order to force a synchronous memory swap, then there is the possibility of time-glitches between swap commands to take into account and other factors), but we think we have found a correct solution for all these problems.

Let's see what has already been built:

- The first component we developed has been an abstract layer to the underlying PCI device driver (our code is written in C++).
  This allows us to swap to other device drivers, besides the Jungo Driver, eventually skipping license fees problems (we could even write our own light-weight driver, we think we know how to do that)

FPGA programming

- The second major component has been the micro-programming of the ALTERA FPGA.
  Key issues here are:

1. Swap of the two memory banks when a full condition is reached
2. ability to generate an interrupt upon a memory full condition
3. I/O activity and synchronization between banks and detector
3. Possibility to generate specific patterns to feed the memory, thus allowing for debugging tests of the system (no need for an actual detector)

Code generated by the Quartus software (the FPGA firmware) can also be uploaded to the PCI board we have in Milano, thus allowing tests and code development to be carried out in parallel

The Interrupt Handler

- When an interrupt is generated, a thread, spawned by the read-out overall process, listens for it and issues the command to swap all other memory banks eventually starting the read-out. This component is rather complex, since it must perform a whole list of checks before, eventually, becoming the master-swapper:

1. only one board at any given time can be allowed to perform this operation, in order to avoid unwanted multiple swaps)
2. care must be taken to avoid commanding a board to swap just after it swapped on it's own because its memory got filled. (the FPGA has been programmed to reject a swap if its read-out pointer has not been reset, and this happens only when ALL memories have been flushed to the overall shared memory)

The Shared Memory

- The heart of the system is a piece of code that creates a shared memory (or attaches to it if it's already present). This shared memory has been implemented as a circular buffer, with all the utilities needed to inspect it's content and synchronize the fill-in and read-out to disk.
- The obvious advantage of such an architecture is that there is a complete logical decoupling between the activity of the PCI board and the activity of the host computer. The timing is NOT dictated by the read-out process but by the detector itself.
- Another advantage is that a process to check the internal consistency of the data flow is just a consumer of this shared memory, but a completely separate and detached process (this component is still missing, though)

The Message Reporter

- In order for all the processes, acting as the DAQ, to synchronize their activity, a light-weight package has been developed to act as as a Message Reporter (based on the IPC protocol).
  More refined packages could also be used, but at this stage we would like to keep the total amount of code to a minimum for better control.

The Code Management

- Since our activity is already split between Milano and Fermilab, we adopted CVS as a code management system since the beginning. Periodic tags and releases are produced on AFS space, making the code available to users in real time.

Preliminary benchmarks (1)

- We have carried out preliminary tests to check the sustainable data flow we can expect. We still have to understand several issues, but we can summarize here the key facts:
- The PLX 9030 PCI Controller on the PTA card we are using does NOT allow for DMA transfer. It could in principle be used in burst-mode, but this feature is not available on the INTEL architecture. We are thus forced to run in single word transfer mode, but this seems more than adequate for our beam test needs.
- We have tried three different setups:

1. A PCI card directly connected to a 1.3 GHz PC motherboard
2. A PCI card directly connected to a 1.8 GHz PC motherboard
3. A PCI card connected to a 1.8 GHz PC motherboard by means of the PCI extender bus

Preliminary benchmarks (2)

- We observe the following data transfer rate from the PCI board to the shared memory in the three cases mentioned above:

	1.3 GHz	1.8 GHz
Direct connection to motherboard	6 Mb/sec	4 Mb/sec
Connection to motherboard by PCI extender	-	2 Mb/sec

- We still do not fully understand these numbers, but studied are under way

To do

- In this overall architecture several components are still missing:
  1. A sophisticated error handling, encompassing both errors generated by the hardware (suppression of extremely noisy channels, dead PCI boards, faulty memories etc...) and by run time conditions (such as excessively high rate due to beam conditions and such)
  2. Detailed benchmark tests to determine the upper limit of the sustainable data rate. We already have made measurements and determined bottlenecks but a systematic work has still to be done.
  3. Try a version of the swapping mechanism where the PCI boards have been modified to accomodate a hardware line to synchronize the interrupt generation.
  4. Other more mundane components, such as run control, activity monitor and logger, GUI, data consistency checker and flow monitor